Parallelization and Performance of the NIM Weather Model for CPU, GPU and MIC Processors

Mark Govett[1], Jim Rosinski[2], Jacques Middlecoff[2], Tom Henderson[2], Jin Lee[1], Alexander MacDonald[1], Paul Madden[3], Julie Schramm[2], Antonio Duarte[3]

Preprint Submitted to BAMS for Review
February 9, 2016

Abstract

The design and performance of the NIM global weather prediction model is described. NIM was designed to run on GPU and MIC processors. It demonstrates efficient parallel performance and scalability to tens of thousands of compute nodes, and has been an effective way to make comparisons between traditional CPU and emerging fine-grain processors. Design of the NIM also serves as a useful guide for fine-grain parallelization of the FV3 and MPAS models, two candidates being considered by the NWS as their next global weather prediction model to replace the operational GFS.

The F2C-ACC compiler, co-developed to support running the NIM on GPUs, has served as an effective vehicle to gain substantial improvements in commercial Fortran OpenACC compilers. Performance results comparing F2C-ACC with commercial GPU compilers demonstrate their increasing maturity and ability to efficiently parallelize and run next generation weather and climate prediction models.

This paper describes the code structure and parallelization of NIM using F2C-ACC, and standards compliant OpenMP and OpenACC directives. NIM uses the directives to support a single, performance-portable code that runs on CPU, GPU and MIC systems. Performance results are compared for four generations of computer chips. Single and multi-node performance and scalability is also shown, along with a cost-benefit comparison based on vendor list prices.

1. Introduction

A new generation of High-Performance Computing (HPC) has emerged, referred to as Massively Parallel Fine Grain (MPFG). The term "Massively Parallel" refers to systems containing tens of thousands to millions of processing cores. "Fine Grain" refers to loop level parallelism that must be exposed in the application to permit thousands to millions of arithmetic operations to be executed every clock cycle. Two general classes of MPFG chips are available: Many Integrated Core (MIC) from Intel and Graphics Processing Units (GPUs) from NVIDIA and AMD. In contrast to up to 36 cores used on the latest generation Intel Haswell CPUs, these MPFG chips contain hundreds to thousands of processing cores. They provide 10-20 times greater peak performance than CPUs, and they appear in systems that increasingly dominate the list of top supercomputers in the world (Top500, 2015). Peak performance does not translate to real application performance however. Good performance can only be achieved if fine-grain parallelism can be found and exploited in the codes. Fortunately, most weather and climate codes contain a high degree of parallelism making them good candidates for MPFG computing.

As a result, research groups worldwide have begun parallelizing their weather and climate prediction models for MPFG processors. The Swiss National Supercomputing Center (CSCS) has done the most comprehensive work so far. They parallelized the dynamical core of the COSMO model for GPUs in

[1] NOAA Earth System Research Laboratory, 325 Broadway, Boulder, Colorado 80305.
[2] Cooperative Institute of Research in the Atmosphere, Colorado State University, Fort Collins, Colorado 80523.
[3] Cooperative Institute for Research in Environmental Sciences, University of Colorado, Boulder, Colorado 80309.

2013 (Fuhrer et al. 2014). At that time, no viable commercial Fortran GPU compilers were available so the code was re-written in C++ to enhance performance and portability. They reported the C++ version gave a 2.9X speedup over the original Fortran code using same generation dual-socket Intel SandyBridge CPU and Kepler K20x GPU chips. Parallelization of model physics in 2014 preserved the original Fortran code by using industry standard OpenACC compiler directives for parallelization (Lapillonne et al. 2014). The entire model, including data assimilation is now running operationally on GPUs at MeteoSwiss.

Most groups primarily focused on parallelization of model dynamics. The German Weather Service (DWD) and Max Planck Institute for Meteorology (MPI-M) developed the ICON dynamical core, which has been parallelized for GPUs. Early work converted the Fortran using NVIDIA specific CUDA-Fortran and OpenCL, demonstrated a 2X speedup over dual socket CPU nodes. The invasive, platform-specific code changes were unacceptable to domain scientists, so current efforts are focused on minimal changes to the original code using OpenACC for parallelization (Sawyer et al. 2014). Another dynamical core, the Non-hydrostatic ICosahedral Atmospheric Model (NICAM), has been parallelized for GPUs, with a reported 7-8X performance speedup comparing 2 NVIDIA K20x GPUs to single older generation, dual socket Intel Westmere CPUs (Yashiro et al. 2014). Other dynamical cores parallelized for the GPU including the Finite-Volume cubed (FV3) model (Nguyen et al. 2013) used in GEOS-5 (Putnam 2011), and the HOMME (Carpenter et al. 2013) have both shown some speedup versus the CPU.

Collectively, these experiences show that porting codes to GPUs can be challenging, but most users have reported speedups over CPUs. Over time, more mature GPU compilers have simplified parallelization, and improved application performance. However, reporting of results has not been uniform and can be misleading. Ideally, comparisons should be made using the same source code, with optimizations applied faithfully to the CPU and GPU, and run on same generation processors. When codes are rewritten, it becomes harder to make fair comparisons as multiple versions must be maintained and optimized. When different generation hardware is used (Eg. 2010 CPUs versus 2013 GPUs), adjustments should be made to normalize reported speedups. Similarly, when comparisons are made with multiple GPUs attached to a single node, further adjustments should be made. Finally, comparisons between a GPU and a single CPU core give impressive speedups of 50-100X but such results are not useful or fair, and require adjustment to factor in use of all cores available on the CPU.

When Intel released its MIC processor, called Knights Corner in 2013, a new influx of researchers began exploring fine-grain computing. Research teams from NCAR's Community Earth System Model (CESM) (Kim et al. 2013), Weather Research and Forecast (WRF) Model (Michalakes et al. 2015), and FV3 parallelized select portions of these codes for the MIC and reported little to no performance gain compared to the CPU. A more comprehensive parallelization from NOAA's Flow Following Finite volume Icosahedral Model (FIM) (Bleck, R. et al., 2015) included both dynamics coupled with GFS physics running on the MIC (Rosinski 2015). Execution of the entire model on the MIC gave no performance benefit compared to the CPU. A common sentiment in these efforts is that porting applications to run on the MIC is easy, but getting good performance can be difficult. With few reports of better MIC performance than the CPU, most believe their efforts will pay off when the next-generation Intel Knights Landing chip arrives in 2016. Further, optimizations targeting the MIC have improved CPU performance, giving immediate benefit for the work.

This paper describes development of the Non-hydrostatic Icosahedral Model (NIM), a model that was designed to exploit MPFG processors. The NIM was initially designed for NVIDIA GPUs in 2009. Since commercial Fortran GPU compilers were not available at that time, the Fortran-to-CUDA Accelerator (F2C-ACC) (Govett et al. 2010) was co-developed with NIM to convert Fortran code into CUDA, a high-level programming language used on NVIDIA GPUs (CUDA, 2015). The F2C-ACC compiler has been the primary compiler used for execution of NIM on NVIDIA GPUs, and has served as a benchmark for evaluation of commercial OpenACC compilers from Cray and PGI. Using the same source code, the NIM was ported to Intel MIC when these processors became available.

The NIM is currently the only weather model that runs on CPU, GPU and MIC processors using a single source code. The dynamics portion of NIM uses OpenMP (CPU & MIC), OpenACC (GPU) and F2C-ACC (GPU) directives for parallelization. For distributed memory parallelization, Scalable Modeling System (SMS) directives are used to handle domain decomposition, inter-process communications, and I/O operations (Govett et al. 1996). Collectively, these directives allow a single source code to be maintained capable of running on CPU, GPU and MIC processors for serial or parallel execution. Further, the NIM has demonstrated efficient parallel performance and scalability to tens of thousands of compute nodes, and has been useful for comparisons between CPU, GPU and MIC processors.

The rest of this paper describes the NIM dynamical core, with some references to work done on model physics. Section 2 describes the computational design of the NIM. Sections 3 and 4 contain details on the parallelization of NIM for Intel MIC and NVIDIA GPU. Section 4 also compares the performance of the F2C-ACC and the PGI GPU compilers. Section 5 highlights performance and scalability in terms of device, node, and multiple nodes. Using these results, a cost-benefit analysis is given based on vendor list prices. The paper concludes with an analysis of the work and final remarks in Sections 6 and 7.

To appear as a side bar

Many-Core and GPU Computing Explained

Many-Core and Graphics Processing Units (GPUs) represent a new class of computing called Massively Parallel Fine-Grain (MPFG). In contrast to CPU chips that have up to 36 cores, these fine-grain processors contain hundreds to thousands of computational cores. Each individual core is slower than a traditional CPU core, but there are many more of them available to execute instructions simultaneously. This has required model calculations to become increasingly fine-grained.

GPU: GPUs are designed for compute-intensive, highly parallel execution. GPUs contain up to five thousand compute cores that execute instructions simultaneously. As a co-processor to the CPU, work is given to the GPU in routines or regions of code called kernels. Loop-level calculations are typically executed in parallel in kernels. The OpenACC programming model designates three levels of parallelism for loop calculations: *gang*, *worker* and *vector* that are mapped to execution threads and blocks on the GPU. Gang parallelism is for coarse grain calculations. Worker level parallelism is fine-grain, where each gang will contain one or more workers. Vector parallelism is for Single Instruction Multiple Data (SIMD) or vector parallelism that is executed on the hardware simultaneously.

MIC: Many Integrated Core (MIC) hardware from Intel also provides the opportunity to exploit more parallelism than traditional CPU architectures. Like GPUs, the clock rate of the chips is 2-5 times slower than current generation CPUs, with higher peak performance provided by additional processing cores, wider vector processing units, and a fused multiply-add (FMA) instruction. The programming model used to express parallelism on MIC hardware is traditional OpenMP threading along with vectorization. User code can be written to offload computationally intensive calculations from the CPU to the MIC (similar to GPU), run in MIC-only mode, or shared between MIC and CPU host.

2. NIM Design

2.1 Model Description

NIM is a multi-scale model, which has been designed, developed, tested, and run globally at 3-km resolution to improve medium-range weather forecasts. The model was designed to explicitly permit convective cloud systems without cumulus parameterizations typically used in models run at coarser scales. In addition, NIM has extended the conventional two-dimensional finite-volume approach into three-dimensional finite-volume solvers designed to improve pressure gradient calculation and orographic precipitation over complex terrain.

NIM uses the following innovations in the model formulation:

- A local coordinate system that remaps a spherical surface to a plane (Lee et al. 2009)
- Horizontal grid point calculations in a loop that allows any point sequence (MacDonald et al. 2011)
- Flux-Corrected Transport formulated on finite-volume operators to maintain conservative and monotonic transport (Lee et al. 2010)
- All differentials evaluated as finite-volume integrals around the cells
- Icosahedral-hexagonal grid optimization (Wang et al. 2011)

The icosahedral-hexagonal grid is a key part of the model. This formulation approximates a sphere with a varying number of hexagons, but always includes twelve pentagons. (Sadourney, R. et el., 1968; Williamson, D., 1971). The key advantage of this formulation is the nearly uniform grid areas that are possible over a sphere as illustrated in Figure 1. This is in contrast to the latitude-longitude models that have dominated global weather and climate prediction for 30 years. The nearly uniform grid represents the poles without the notorious "pole problem" inherent in latitude-longitude grids where meridians converge toward the poles.

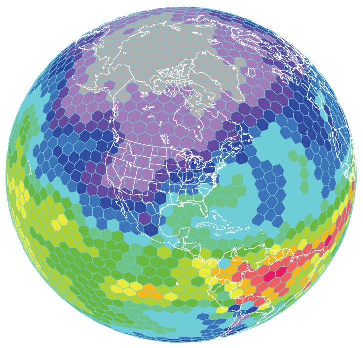

Figure 1: Icosahedral-hexagonal grid at approximately 450 km horizontal resolution.

NIM uses a fully three-dimensional finite-volume discretization scheme designed to improve pressure gradient calculations over complex terrain. Three-dimensional finite-volume operators also provide accurate and efficient tracer transport essential for next-generation global atmospheric models. Prognostic variables are co-located at horizontal cell centers (Arakawa, A., and V. Lamb, 1977). This simplifies looping constructs and data dependencies in the code.

The numerical scheme utilizes a local coordinate system remapped from the spherical surface to a plane at each grid cell. All differentials are evaluated as finite-volume integrals around each grid cell. Flux corrected transport is formulated on finite-volume operators to maintain conservative positive-definite transport. Passive tracers are strictly conserved to the round-off limit of single-precision floating-point operations. NIM governing equations are cast in conservative flux forms with mass flux to transport both momentum and tracer variables.

2.1 Computational Design

NIM is a Fortran code containing a mix of Fortran 77 and Fortran 90 language constructs. It does not use derived types, pointers or other constructs that can be challenging for compilers to support or run efficiently[4]. The SMS library utilized by NIM for coarse-grain parallelism employs the MPI (Message Passing Interface) library to handle domain decomposition, inter-process communications, reductions, and other MPI operations.

NIM was designed from the outset to maximize fine-grain or loop-level parallel computational capability of both NVIDIA GPU and Intel MIC architectures. Primary model computations are organized as simple dot products or vector operations, and loops with no data-dependent conditionals or branching. The NIM dynamical core requires only single-precision floating point computations and runs well on a single CPU core, achieving 11% of peak performance on a single Intel SandyBridge CPU core.

Grid cells can be stored in any order. A lookup table is used to access neighboring grid cells and edges on the icosahedral-hexagonal grid. The model's loop and array structures are organized with the vertical dimension innermost in dynamics routines. Testing during model development verified that the indirect scheme yielded less than 1 percent performance penalty because the cost of the extra reference can be amortized over the innermost, directly addressed, vertical dimension (MacDonald et al. 2010). Further, unit-stride accesses in the vertical dimension support CPU and MIC inner-loop vectorization, and efficient memory accesses on the GPU.

For the GPU, NIM dynamics has been parallelized with F2C-ACC and OpenACC directives and executes completely on the GPU. Model state remains resident in GPU global memory. In the model dynamics, data are only copied between CPU and GPU (across the PCIe bus) for disk I/O and between GPUs for inter-process communications. Physical parameterizations have not yet been ported to the GPU, so data must be moved between the GPU and CPU every physics time step. GPU to GPU communications is handled via SMS directives and initiated by the CPU.

Fine-grain parallelization for MIC (and also multi-core CPU nodes) is implemented via OpenMP directives. MIC parallelization of NIM was easy since the code had already been modified to run efficiently on the CPU and GPU. Symmetric execution, where both the CPU host and attached MIC co-processor are used, is also permitted and will be described in Section 3.

Example 1 shows representative code from the NIM dynamics with OpenACC, F2C-ACC and OpenMP directives prefaced with !$acc, !ACC$, and !$OMP respectively. F2C-ACC does not adhere to the OpenACC standard, though the directives and their placement in source code are quite similar. In this example, the OpenMP threaded region is identified by !$OMP PARALLEL DO, which corresponds to !ACC$REGION and !$acc parallel that are used to define GPU kernels. F2C-ACC directives will be removed from NIM once ongoing comparisons with OpenACC compilers are complete.

[4] The OpenACC specification only recently added support for derived types; pointer abstractions may limit the ability of compilers to fully analyze and optimize calculations.

```
    !$OMP PARALLEL DO PRIVATE (k)
    !ACC$REGION(<96>,<10242>) BEGIN
    !$acc parallel num_gangs(10242) vector_length(96)

    !ACC$DO PARALLEL(1)
    !$acc loop gang
    do ipn=ips,ipe       ! Loop over horizontal

    !ACC$DO VECTOR(1,1:nz-1)
    !$acc loop vector
      do k=1,nz-1        ! Loop over vertical levels
       bedgvar(k,ipn,1) = . . .
       bedgvar(k,ipn,2) = . . .
     end do

         < more loops and calculations >
    !$acc parallel end
   !ACC$REGION END
   !$OMP END PARALLEL DO
```

Example 1: Representative Fortran code from NIM, parallelized with OpenMP, F2C-ACC and OpenACC directives. For clarity in this example, numeric arguments for the number of threads (96) and blocks (10242) in the directive are used that equate to the loop bounds `nz` and `ipe-ips+1` respectively. In the actual NIM code, variable arguments are used.

3. CPU and MIC Parallelization

Parallelization for the CPU and MIC involved three steps: (1) insertion of OpenMP directives to identify thread-level parallelism in the code and fusing multiple threaded loops where possible, (2) vectorization optimizations using directives and compiler flags, and (3) other code optimizations and runtime configurations to improve performance.

3.1 Vectorization

Vectorization is an optimization where independent calculations executed serially within a loop can be executed simultaneously in hardware by specially designated vector registers available to each processing core. The number of operations that can be executed simultaneously is based on the length of the vector registers. On the CPU, vector registers are currently 256 bits in length; the KNC MIC co-processor contains 512-bit vector registers. Based on these vector lengths, eight single-precision operations can be executed simultaneously on the CPU, and sixteen such simultaneous operations on the MIC. As a result, vectorizing loops provided some benefit on the host, but in most cases it provided a greater improvement on the MIC. Intel compilers automatically attempt vectorization by default, with compiler flags available for further optimization on specific hardware.

3.2 Optimizations

OpenMP parallelization of NIM dynamics for both host and MIC is done exclusively at the horizontal loop level. Horizontal looping in almost all cases is outside of vertical looping and, if applicable, loops over cell edges. Indirect addressing of horizontal indices (described earlier) allows for a single loop over horizontal grid points owned by each MPI task. This single loop structure enables a great deal of thread parallelism, without relying on restrictive OpenMP verbs such as "*collapse*" (fuses multiple OMP loops into one), which are required in some codes to expose sufficient parallelism. In NIM there are no MPI communications inside of threaded loops. This enables use of a "thread funneled" MPI library (most MPI libraries are built this way), where the assumption that no MPI communication happens inside of threaded loops enables the MPI library to avoid numerous performance-sapping locks and tests on threading.

Most OpenMP loops in NIM contain sufficient work to easily amortize the cost of assigning work to individual threads on loop startup, and thread synchronization at the end of the loop. These costs are significantly higher on the MIC than on the SNB host. Part of the reason being that the MIC has more threads which must be synchronized, with as many as 244 per MIC card vs. 16 on the SNB host. Also the clock cycle on the MIC is more than twice as slow as that of the SNB host. As a result, fusing of some threaded loops gave a bigger performance boost on the MIC than the host. Some boost was visible on both architectures because in each case the number of thread synchronization points was reduced.

The OpenMP standard includes a verb ("guided") which tells the OpenMP runtime library to parcel out iterations of the subsequent threaded loop in a more sophisticated way than the default mechanism ("static"), where each thread is handed an equal number of loop iterations. When the "guided" verb is in force, the OpenMP runtime library parcels out initially relatively large numbers of loop iterations to threads. Subsequent iterations are parceled out in "hungry puppy" fashion to threads which have finished their initial work, with the size of each iteration set diminishing as work done by the threads progresses. This procedure has a positive impact on load imbalance. Though most horizontal loops in the NIM dynamics contain equal work per loop iteration, it was found that the "guided" verb provided measurable benefit in some cases on the MIC. Possible causes for this imbalance, even given equal work load, include memory contention, cache utilization, and OS intervention.

Though the discussion in this paper is focused almost exclusively on the NIM dynamics, work on the physics package utilized in NIM has shown a great advantage in deploying the "guided" verb on physics horizontal loops exercised on both CPU and MIC. In NIM the physics loops are both vectorized and threaded over horizontal iterations. Each thread is provided at least enough horizontal points at a time to enable vectorization of all the points it is given. This "chunking" approach in a single dimension is necessary because most of the physical parameterizations contain dependencies in the vertical dimension that precludes most opportunities to vectorize and/or thread. The parameterizations contain much inherent load imbalance, such as solar radiation calculations only being done in sunlit regions, and cloud calculations only necessary where clouds are present. Thus the "guided" verb is ideal for many of the physical parameterizations. NIM contains a single horizontal loop over all physical parameterizations, so "guided" provides enormous benefit on both the SNB and the MIC.

3.3 Symmetric Execution

Parallelization of NIM for the MIC was done using symmetric mode, where both the host and attached MIC co-processor share the workload. Communication between tasks is accomplished using the same MPI primitives as homogeneous (CPU-only) mode. The only difference is that separate compilations are required for the different (CPU and MIC) architectures. Complications imposed by various communication patterns in a multi-node run (e.g. host-host, host-mic, mic-mic) are blissfully hidden from the application developer by the MPI implementation.

In a heterogeneous computational environment (in this case host and MIC), inevitably there will be load imbalances resulting from the different computational capabilities of the separate architectures. Noting that the SMS library attempts to equally distribute the number of horizontal points assigned to each MPI task, an attempt to mitigate this imbalance was made in NIM by varying the number of MPI tasks placed on each device. The "stampede" system at TACC (Texas Advanced Computing Center) was used for most of the MIC porting activity with NIM. This system employs SandyBridge (SNB) host processors and KNC MIC co-processors. On this system it turned out that the time to run a single NIM time step on a host node roughly equaled the time to run that same time step on a MIC card. Thus, placing an equal number of MPI tasks on the host(s) and attached co-processor(s) worked well.

Obtaining rough equivalent performance on a single SNB node and the attached MIC card required some software engineering work. After the initial port, the SNB node was substantially faster than the MIC. Subsequent code modifications to address vectorization and optimizing OpenMP performance brought the two architectures into balance. Obtaining balance was more difficult than originally anticipated, mainly because code modifications designed to improve performance on the MIC also improved performance on the host.

4. GPU Parallelization

4.1 OpenACC Compiler Evaluation

Either OpenACC or F2C-ACC directives can be used for GPU parallelization of NIM. F2C-ACC has been an effective way to push for improvements in commercial Fortran GPU compilers. Prior evaluation of OpenACC compilers and their predecessors was done in 2011 (CAPS, PGI) (Henderson et al. 2011) , 2013 (PGI, Cray) (Govett, 2013) and in 2014 (PGI, Cray) (Govett et al. 2014). These evaluations exposed bugs in the compilers, and areas where additional development was needed to support weather prediction models used at NOAA. A comprehensive performance evaluation in 2014 showed the Cray and PGI compilers ran the dynamics portion of NIM 1.7 and 2.1 times slower than F2C-ACC. This led to increased collaboration between NOAA and vendors to improve the performance of their respective compilers.

Table 1 shows the runtimes for NIM dynamics routines with near parity observed in performance with the F2C-ACC and PGI compilers. PGI improvements since the 2014 evaluation can be attributed to: (1) better support user-defined fast memory, (2) direct mapping of do-loops when loop bounds are known and do not exceed GPU hardware limits, and (3) performance optimizations that were introduced or improved. While most routines run faster with the PGI compiler than F2C-ACC, two routines (vdmints and vdmintv) are slower. The cause is under investigation, but higher register usage by the PGI compiler may explain the performance degradation[5].

Routine	F2C-ACC v5.8	PGI version 15.10	PGI Speedup
vdmints	4.58	5.40	0.84
vdmintv	1.63	2.29	0.71
diag	1.24	0.65	1.90
flux	0.98	1.03	0.95
force	0.61	0.52	1.17
trisol	0.36	0.28	1.28
timediff	0.18	0.15	1.20
Total Runtime	**11.77**	**12.34**	**0.95**

Table 1: Performance of the NIM dynamics with the F2C-ACC, PGI and Cray compilers. Runtimes in seconds for main routines are shown. Speedup represents runtimes compared to F2C-ACC. Vdmints runtime represents the sum of three variants in the model: vdmints, vdmints0, and vdmints3.

While performance with both compilers is similar, code parallelization is simpler with the OpenACC compilers, which speeds the time required to port applications to the GPU. The rest of this section describes OpenACC parallelization of NIM, with some references to F2C-ACC capabilities that led to improvement in the Cray and PGI compilers.

4.2 OpenACC Parallelization

Parallelization for the GPU can be expressed in three phases: defining GPU kernels and identifying loop-level parallelism, minimizing data movement, and optimizing performance. Parallelization of NIM

[5] GPUs have a limited number of registers in hardware. Excessive register use can lead to lower occupancy or concurrency that limits application performance.

for these phases will be described in the next three sections. For brevity, the sentinels (eg. !$acc) are omitted.

4.2.1 Defining GPU Kernels and Parallelism

OpenACC's **parallel** directive is used to define code sections, called kernels, that will be executed on the GPU.[6] The number of threads and blocks used (96 and 10242 respectively in Example 1) are required arguments for F2C-ACC that map directly to CUDA thread and grid blocks in the generated code. While no arguments are required with **parallel**, it was best to specify resources explicitly using *num_gangs* and *vector_length* (see Example 1) to get optimal performance. A vector length of 96 was used that matched the number of vertical levels in the model. OpenACC's **loop** directive is used to identify parallel loops along with *vector* and *gang* keywords. Using similar directives, F2C-ACC demonstrated the value of directly mapping loops to thread and grid block execution on the GPU. This led to improvements to the PGI compiler to provide similar capabilities.

4.2.2 Minimizing Data Movement

By default, OpenACC compilers assume data are resident on the CPU and automatically generate data copies between the CPU and GPU necessary to execute each kernel. However, to run efficiently, data movement between the CPU and GPU must be minimized by keeping data resident on the GPU. With OpenACC, data regions are defined using the **data** directive with data movement specified optionally by listing variables to be managed using the *pcopy*, *pcopyin*, and *pcopyout* keywords. The letter "p" preceding the copy is shorthand for "*present_or*" and indicates the runtime system check to determine when a data transfer is needed. Asking the runtime system to always perform the check incurs little overhead. To keep data resident on the GPU, variables are listed in the **data** directive with *enter* and *exit* clauses indicate data creation and removal.

To further simplify data management with OpenACC compilers, the CUDA runtime library supports unified memory, a means to programmatically treat CPU and GPU memory as one large memory. With unified memory, data are automatically copied between GPU and CPU on demand by the runtime system. While less efficient than when users prescribe data transfers themselves, it reduces the time and effort required to get applications running on the GPU. Using unified memory, data movement between CPU and GPU becomes a performance optimization rather than a debugging challenge.

To date, unified memory is managed in software by the NVIDIA runtime system. With the next generation chip (called "Pascal"), data accesses will be managed in hardware. The hardware will include a new high-speed interconnect called NVLINK, that will allow the GPU to access CPU memory at CPU memory speeds. This will allow CPU and GPU memory to be treated as one common fast memory, simplifying GPU programming and improving application performance.

4.2.3 Optimizing Performance

Fine-tuning performance primarily means optimizing the use of the multi-tiered memory available on the GPU. To reduce memory use, F2C-ACC supports variable demotion to remove a dimension from a locally defined variable. Demotion is particularly beneficial when a single dimensional array can be replaced with a scalar. This transformation, called scalar replacement, improves performance by replacing slower global memory access with faster register accesses.

OpenACC memory access is intentionally more restrictive, to avoid reference to specific details of the underlying hardware. This improves portability and leaves the compilers free to implement efficient use of memory where possible through compiler analysis and optimization. Implicit use of faster local and fast shared memory is possible in OpenACC using the *private* clause attached to gang or vector loops. In addition, improved support for **cache** memory by the Cray and PGI compilers permits explicit

[6] OpenACC's **kernel** directive was also tried. While simpler to use, it provided less user control over parallelization, and slower performance than **parallel**.

user-managed shared memory to improve application performance on the GPU. Variable demotion or scalar replacement is not exposed in the OpenACC standard, but is handled automatically when analysis determines such optimizations can be done.

5. Performance and Scaling

The NIM code has been highly optimized for serial and parallel execution. It has demonstrated good scaling on both CPUs and GPUs on Titan[7] where it has run on more than 250,000 CPU cores and more than 15,000 GPUs. It has also been run on up to 320 Intel MIC (Xeon-Phi) processors at the Texas Advanced Computing Center (TACC)[8]. Optimizations targeting Xeon-Phi and GPU have also improved CPU performance.

Since NIM has been optimized for the CPU, GPU and MIC, it is a useful way to make comparisons between chips. Every attempt is made to make fair comparisons between same generation hardware, using identical source code optimized for all architectures. Given the increasing diversity of hardware solutions, results are shown in terms of device, node and multi-node performance.

5.1 Device Performance

Single device performance is the simplest and most direct comparison of chip technologies. Figure 2 shows performance running the entire NIM dynamical core on four generations of CPU, GPU and MIC hardware. CPU results are based on standard two socket node configurations. A roughly 2X performance benefit favoring accelerators is observed for 2010 through 2014 generation GPU chips. In addition, the MIC Knights Corner processor demonstrates a 1.3X speedup over the IvyBridge CPU.

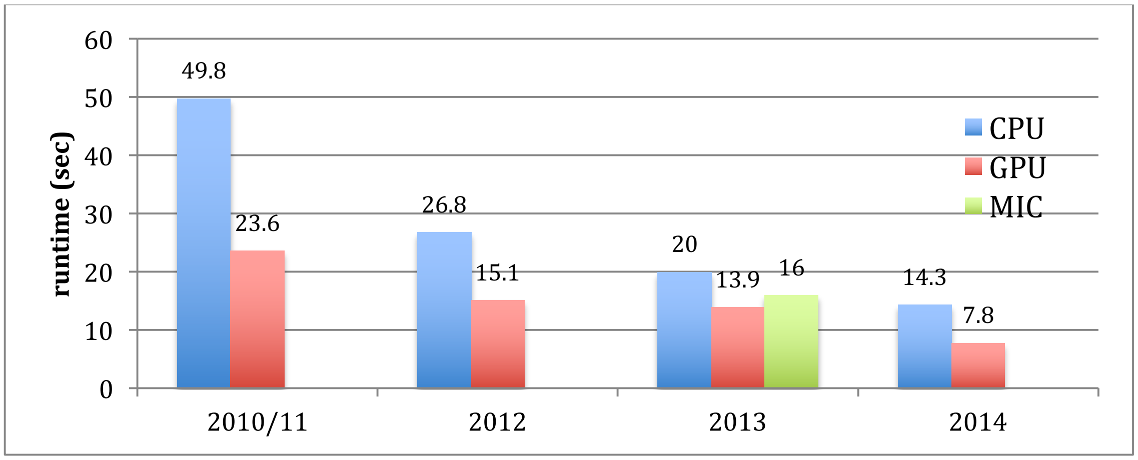

Figure 2: Runtimes for the NIM running at 240 km resolution (10242 horizontal points, 96 vertical levels) for 100 time steps with the following chips[9]:

Year	CPU: 2 sockets	Cores	GPU	Cores	MIC	cores
2010/11	Westmere	12	Fermi	448		
2012	SandyBridge	16	Kepler K20x	2688		
2013	IvyBridge	20	Kepler K40	2880	Knights Corner	61
2014	Haswell	24	Kepler K80	4992		

[7] Titan is an AMD-GPU based system containing over 17,000 GPUs, managed by the U.S. Department of Energy's Oak Ridge National Laboratory (ORNL).

[8] TACC is a National Science Foundation (NSF) supported CPU-MIC cluster at the University of Texas, Austin.

[9] Nodes configured with 2 sockets/node were: Westmere: x5650, SandyBridge: E5-2670, IvyBridge E5-2690V2, Haswell E5-2690V3

Improvements to the CPU chips, including increased memory bandwidth, advanced vector instructions, and steadily increasing core counts, have been sufficient to generally keep pace with accelerators. Of particular importance in the CPU results is the nearly 1.9X decrease in runtime for the SandyBridge over the Westmere processor (26.8 and 49.8 seconds resp.), largely due to a 2X increase in memory speed. This is because weather and climate dynamical cores are usually memory bound, rather than compute bound.

Hyperthreading is a technique where the number of execution threads is greater than the number of available processing cores. Overscheduling can be an effective optimization specified trivially at run-time for CPU, GPU and MIC. Scaling results are shown for a single MIC processor in Figure 3. The MIC processor contains 61 physical cores, increasing performance demonstrated out to the maximum of 244 threads permitted on the device. Hyperthreading can be beneficial when a thread running on a core is stalled waiting on a memory read or write. In this case the operating system can swap out the running thread and swap in a waiting thread that is not otherwise waiting on hardware resources. Near linear speedup to the 61 physical cores is observed, with 4-way hyperthreading providing nearly an additional 2X performance advantage beyond that.

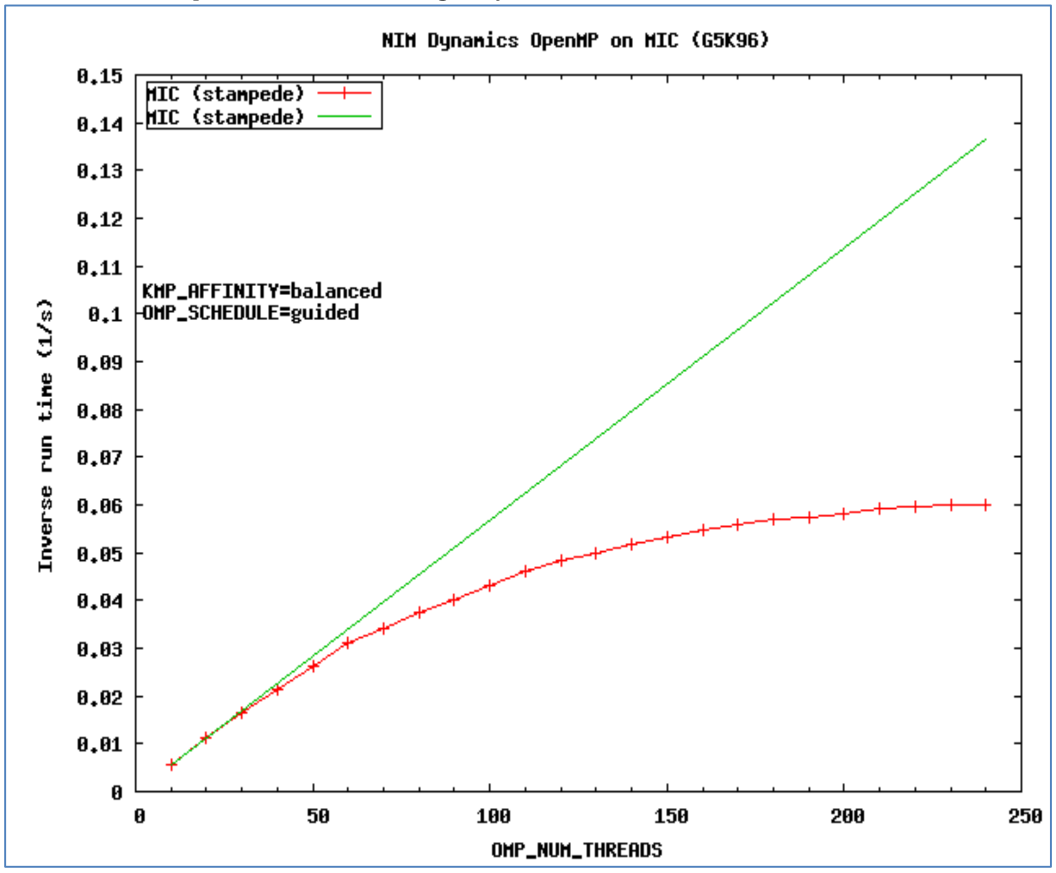

Figure 3: NIM thread scaling on a single MIC processor containing 61 cores. The green line is linear speedup reference; the red curve is observed speedup. NIM dynamics achieves nearly linear scaling up to 61 threads, with a 2X further speedup when hyperthreading with up to 244 virtual threads.

Since accelerators currently must be attached to a CPU host, price-performance benefit is reduced when the cost of all hardware is included. This practical and economic consideration motivates comparisons between nodes that include both the host and attached accelerators.

5.2 Single Node Performance

The basic components of Intel CPUs include two CPU sockets, memory, network interconnect (NIC), Peripheral Component Interconnect express (PCIe) bus and a motherboard. Deviations from this basic configuration are available but more expensive since the volumes manufactured are lower. Therefore, most computing centers use standard, high volume parts that offer the best price-performance. Accelerators are attached to these nodes and communicate with the CPU host via the PCIe bus. Figure 4 shows a simple illustration of two CPU nodes, each containing two attached accelerators and connected via high-speed network interconnect.

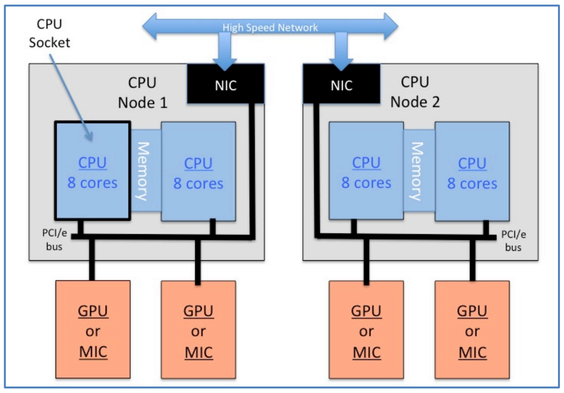

Figure 4: Standard Intel Sandy Bridge node configuration containing two CPU sockets per node with attached accelerators.

Figure 5 shows performance of the NIM dynamical core for several traditional CPU and accelerator configurations. Standard two socket Intel IvyBridge nodes were used with attached MIC and GPU accelerators. The four left-most results show runtimes when only the CPU, GPU and MIC are used and appear individually in blue, green and orange respectively. The shaded results highlight performance of the NIM for symmetric execution, where both the CPU and attached accelerator are used. The two runtimes appear for each configuration with symmetric runs, one in blue and a second in green or orange, with a numeric value representing the node runtime at the top of each column.

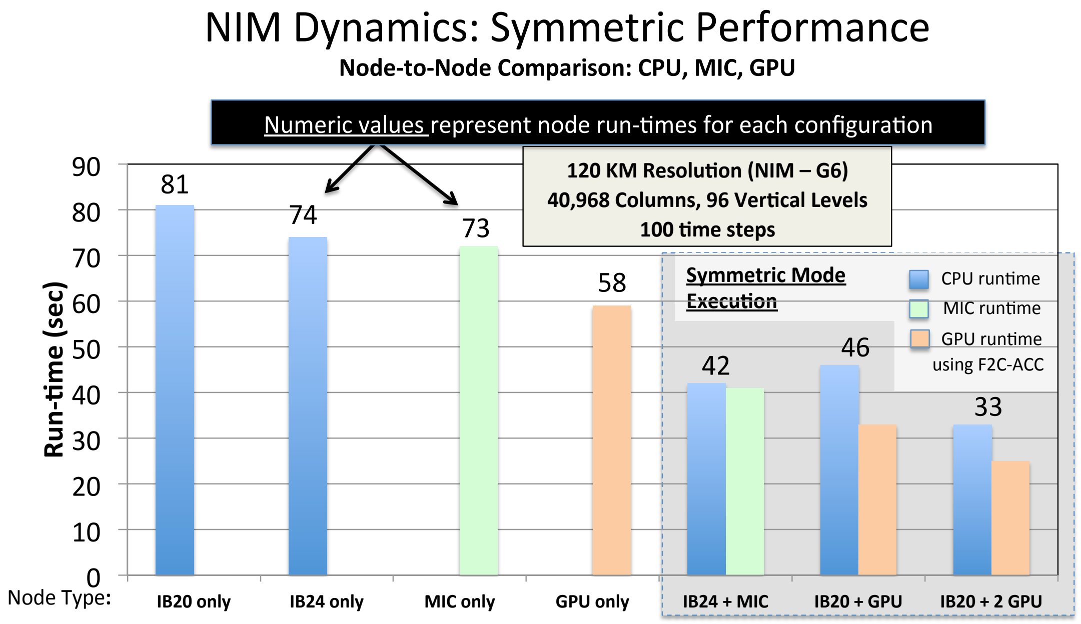

Figure 5: Full node performance for CPU, GPU and MIC using standard two socket Intel IvyBridge CPUs and one or two attached accelerators. Symmetric mode execution gives performance when both the CPU and GPU or MIC are used. Runtimes for the NIM running at 120 km resolution (40968 columns, 96 vertical levels) for 100 time steps with the following hardware:

IB20 – IvyBridge,	20 cores, 3.0 GHz (E5 2690-v2)	**GPU** - Kepler K40	2880 cores, 0.745 GHz
IB24 – IvyBridge,	24 cores, 2.6 GHz (E5 2697-v2)	**MIC** – KNC 7120	61 cores, 1.238 GHz

For symmetric runs, the model sub-domain is divided evenly between the host and device leading to differences in execution time as shown. When the accelerator is faster than the host, it must wait for the host to complete. This explains the slower overall node time of 46 seconds for the CPU-GPU configuration (IVB20-GPU), where the GPU runtime is significantly faster than the host (33 versus 46). Performance of the 24 core IvyBridge was nearly equivalent to the MIC (KNC), leading to a balanced symmetric result.

These results show significant performance benefit using symmetric execution, especially when the number of accelerators is small. Comparisons using two accelerators were tested that gave continued improvement in runtimes for the GPU, but worse performance for the MIC (not shown) due to hardware resource limitations. Given Intel's plans to develop a host-less MIC chip (Knights Landing), there was little reason to pursue resolution of this issue. On the GPU, symmetric execution yields diminishing benefit as the number of accelerators attached to a single node increases. Since eight or more GPUs can be attached to a single node, performance is further examined in terms of strong and weak scaling.

5.2.1 Strong Scaling

Strong scaling is measured by applying increasing numbers of compute resources for a fixed problem size. This metric is particularly important for operational weather prediction where forecasts should run in one-percent of real-time. The requirement is normally achieved by increasing the number of processors until the given time threshold is met. For example, a one-day forecast that runs in 15 minutes represents one percent of real-time; runs in two percent of real-time would take 30 minutes.

Figure 6 shows strong scaling for NIM running at 120KM resolution on a single CPU containing 1 to 8 GPUs per node. To reduce inter-GPU communications, NVIDIA developed GPUdirect, a means of bypassing the CPU host in MPI data transfers. Use of GPUdirect improved multi-GPU runtimes for the NIM by up to 38 percent on a single node (Middlecoff, 2015). In the figure, "cols/GPU" is the number of vertical columns assigned to a single GPU. The term column refers to a single icosahedral grid cell with 96 vertical levels. As the number of GPUs increase from 1 to 8, efficiency decreases because there are fewer columns and thus fewer calculations per GPU. These strong scaling results show the best efficiency (> 90%) is achieved when at least 10,000 columns of work are given to each GPU.

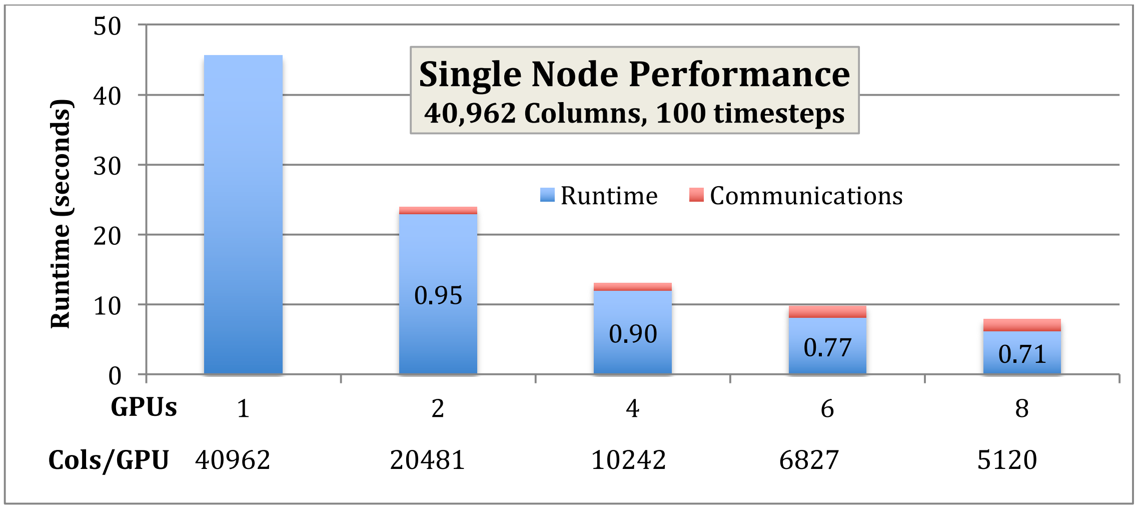

Figure 6: Strong scaling performance, where "Cols/GPU" indicate the amount of work (columns per GPU) each device must do. Numeric values represent speedup over a single GPU. NVIDIA K80s packaged with 2 GPUs per board were used for these results, so the 2 GPU result ran on one K80.

5.2.2 Weak Scaling

Weak scaling is a measure of how solution time varies with increasing numbers of processors, when the problem size per processor is fixed. It is considered a good way to determine how a model scales to high numbers of processors and is particularly useful for measuring communications overhead. Table 2 gives performance results for a single node with 20,284 columns per GPU for 120KM and 60KM resolution runs using 2 and 8 GPUs. NVIDIA K80s packaged with two GPUs were used for the runs. Computation time is nearly identical for all runs, with communications time increasing to 3.19 seconds for the one node, 8 GPU run. An additional run using two nodes illustrates the substantial increase in off node communications time. Given communications time within a node (3.19 seconds) is less than the off-node time (7.23 seconds), the results show that more GPUs could be added to each node without adversely affecting model runtimes. This is because all processes must wait for the slowest communication to complete before model execution can continue.

GPUs per Node	Number of Nodes	Model Resolution	Columns/GPU	Computation Time	Communications Time	Total Runtime
2	1	120 KM	20,482	25.13	0.56	25.71
8	1	60 KM	20,482	25.16	3.19	28.35
2	4	60 KM	20,482	25.22	7.23	33.45

Table 2: Weak Scaling performance with GPUdirect for GPU to GPU data transfers for a single node (not shaded) and multiple nodes (shaded). Communications time have much higher overhead compared to within node data transfers.

5.2.3 Architectural Considerations

A standard Intel two socket node is normally configured to support up to two accelerators that can communicate with the host at full speed. When more than two devices are attached to the host, they must share the PCIe bandwidth. More specialized solutions are available that reduce this constraint by adding additional PCIe hardware. Figure 7 illustrates the architecture for a Cray-Storm node, with 8 attached accelerators (GPUs are shown but MIC processors can also be used). In this configuration, two accelerators are attached to each PCIe communications hub, with two hubs per CPU socket. Communications between sockets are handled with Intel's Quick Path Interconnect (QPI).

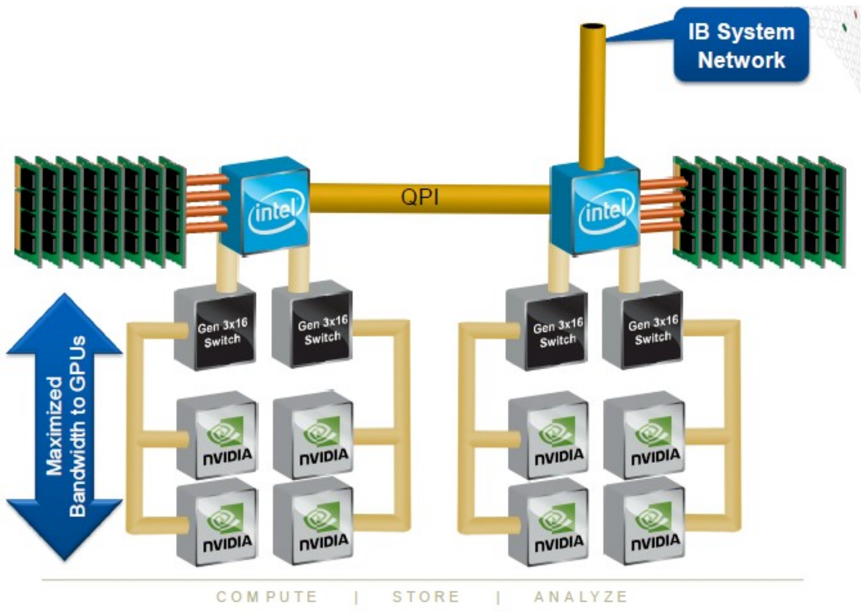

Figure 7: Illustration of the Cray Storm node architecture containing eight accelerators per node. NVIDIA GPUs are shown, but other PCIe compatible devices can be used. Reference from Cray.

One potential performance bottleneck for this node architecture may be the limited bandwidth of the single InfiniBand (IB) connection shared by the eight attached accelerators. Alternative node configurations are available including ones with multiple IB connections, nested PCIe architectures, and solutions that avoid use of QPI, due to reported latency issues (Ellis 2015; Cirrascale 2015).

These more specialized solutions offer potential benefit over standard high-volume parts, where application testing is the best way to determine overall cost-benefit.

5.3 Multi-node Performance and Scaling

Good multi-node performance and scaling on hundreds to thousands of nodes require efficient inter-process communications. For most models, communications normally includes gathering and packing data to be sent to neighboring processes, MPI communications of the data, and then unpacking and distributing the data. Analysis of NIM dynamics performance showed that message packing and unpacking accounted for 50% of inter-GPU communications time. Since NIM relies on a lookup table to reference horizontal grid points, data can be reorganized in any order to eliminate packing and unpacking. This optimization is called "spiral grid order".

Figure 8 illustrates the spiral grid ordering used in NIM. Optimal spiral grid ordering is based on the number of MPI ranks that will be used, and is determined during model initialization. As shown in the figure, points are organized according to where they must be sent (as interior points) or received (as halo points). Each point in the figure represents an icosahedral grid column that contains 96 vertical levels. In the figure, the Spiral Grid Ordering section illustrates the method used to order points within each MPI task. The Data Layout section of the figure illustrates how grid points are organized in memory for optimal communications and computation. Use of the spiral grid order gave performance benefit on all architectures, with a 20 percent improvement in model runtimes on the GPU, 16 percent on the MIC-CPU, and 5 percent on the CPU.

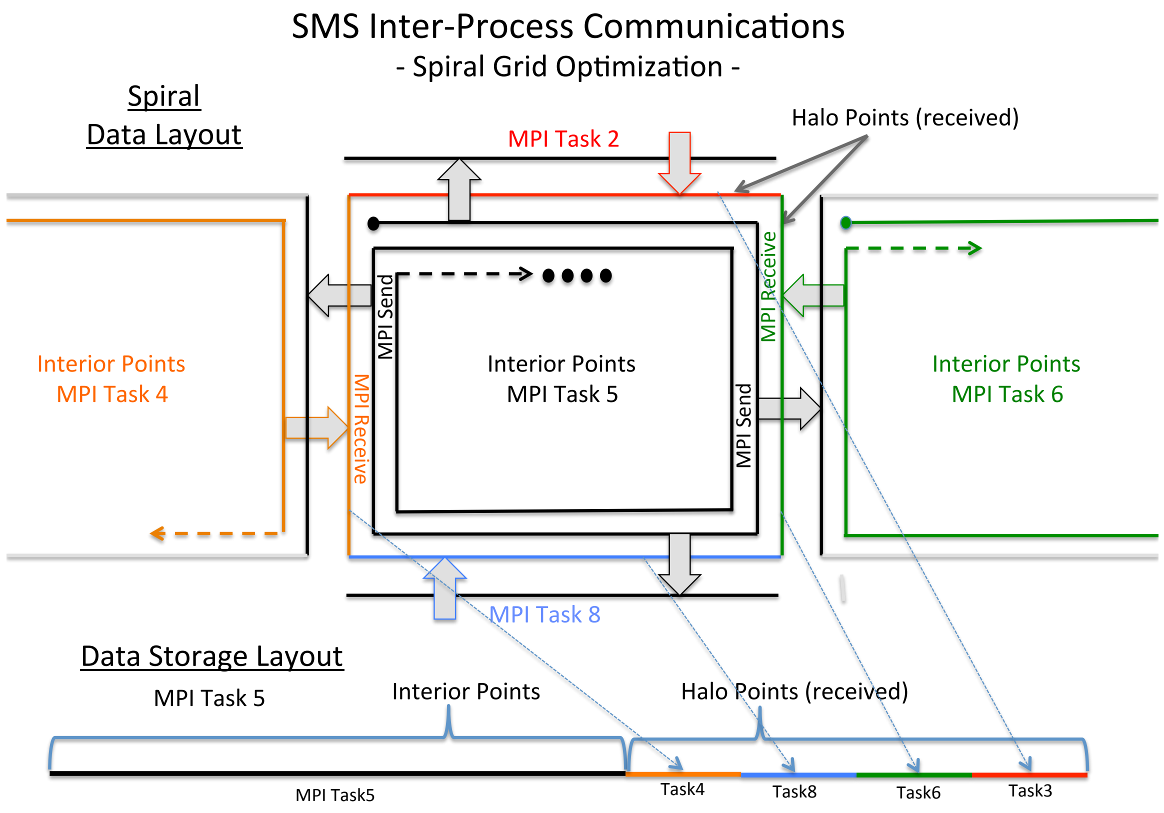

Figure 8: Spiral Communications where **Spiral Data Layout** shows interior points for several MPI tasks, organized with data sent to neighbor tasks in black, and points to be received in different colors from each MPI task. **Data Storage Layout** shows how data are organized in memory, where each horizontal point shown represents a model column containing 96 vertical levels.

Figure 9 shows multi-node scaling results for 20 to 320 CPU, GPU, and symmetric runs with the CPU and MIC. The hardware used in the runs was older generation Intel SandyBridge (2012), NVIDIA K20x (2012), and Intel Knights Corner (2013). These results show the MIC and GPU are best when they have more than 10,000 columns of work per device. CPU performance is best when there is less work per CPU. Increasing efficiency (0.65 to 0.71) and superscalar performance is observed (4096 and 2048 columns resp.) because computations fit better in cache memory.

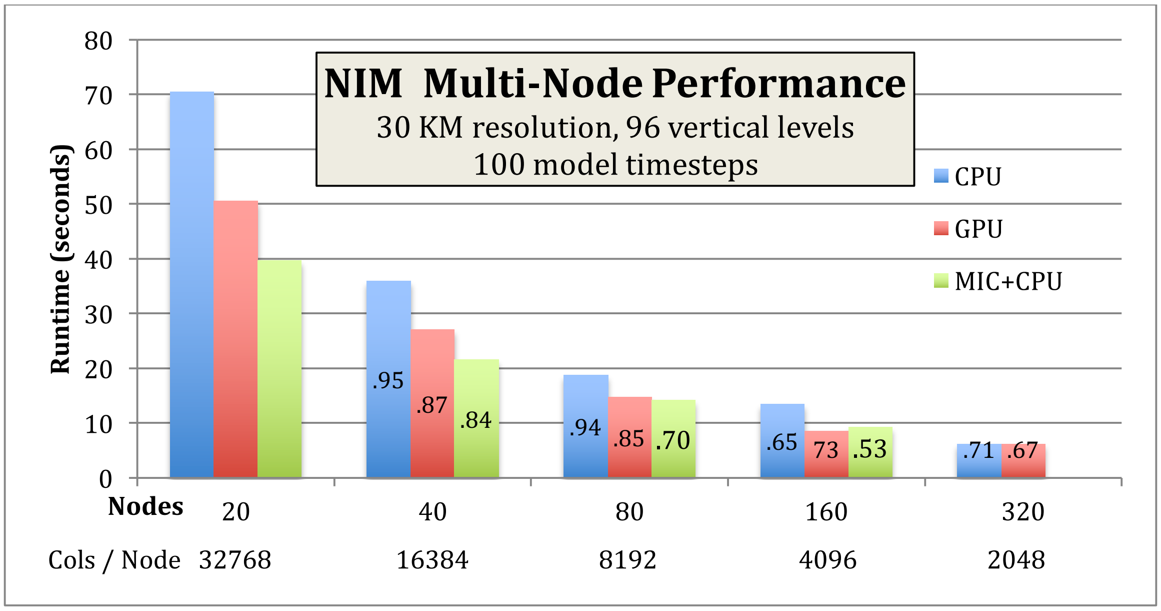

Figure 9: NIM scaling comparison for dual-socket SandyBridge CPU, NVIDIA Kepler K20x GPU, and symmetric execution with Intel Knights Corner (MIC) and a dual socket Intel SandyBridge CPU. The horizontal axis gives the number of nodes used for the fixed problem size. Cols / Node indicate the number of columns per node, consisting of either the dual socket CPU, GPU or split between the CPU and MIC for symmetric execution. Efficiency compared to the twenty node runtime appears as a numeric value in each performance bar.

5.4 Cost-Benefit (GPU)

Cost benefit for the GPU is determined using list prices as specified from Intel and NVIDIA in Table 3. The CPU node estimate was based on a standard two socket, 24 core, Intel Haswell node, that includes the processor, memory, network interconnect, and warranty. The system inter-connect was not included in cost calculations, based on the assumption that the cost for each system would be similar. While significant discounts are normally offered to customers, it would be impossible to fairly represent them in any cost-benefit evaluation here.

Chip	Part	Cores	Power (watts)	OEM Price
Haswell	E5-2690-V3 (2)	24	270	4180[10]
NVIDIA K80	K80	4992	300	5000[11]
Intel MIC (KNC)	7120P	61	300	4129[12]
Haswell CPU Node	Dell R430	24	-	6500[13]

Table 3: List prices for Intel Haswell CPU, Intel MIC, and NVIDIA K80 GPU processors. The CPU node is based on the cost of a Dell R430 rack-mounted system.

[10] http://ark.intel.com/products/81713/Intel-Xeon-Processor-E5-2690-v3-30M-Cache-2_60-GHz
[11] http://www.anandtech.com/show/8729/nvidia-launches-tesla-k80-gk210-gpu
[12] http://ark.intel.com/products/75799/Intel-Xeon-Phi-Coprocessor-7120P-16GB-1_238-GHz-61-core
[13] Dell PowerEdge R430 server, rack mounted, quote 3/2/2015, see appendix for details.

Figure 10 shows a cost-benefit based on running NIM dynamics at 30km model resolution. Each of the five system configurations shown produced a 3 hour forecast in 23 seconds or 0.20 percent of real-time. The CPU-only configuration (upper-left point) required 960 cores or 40 Haswell nodes. The right-most configurations used 20 NVIDIA K80 GPUs that were attached to 20, 10, 8 and 5 CPUs respectively. The execution time of 23 seconds can be extrapolated to 1.6 percent of real-time for a 3.75KM resolution model when per-process workload remains fixed (weak-scaling)[14]. The 20,482 columns of work per GPU, is also consistent with 95 percent device efficiency given in Figure 5.

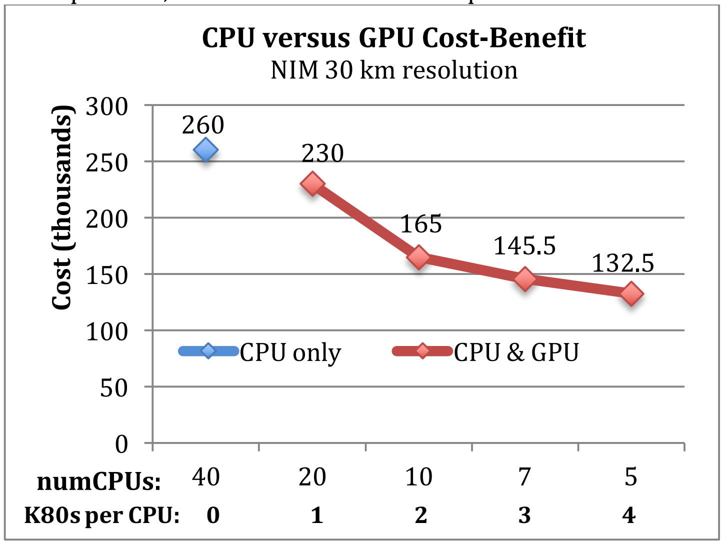

Figure 10: Cost comparison for CPU-only, and CPU-GPU systems needed to run 100 time steps of NIM dynamics in 23 seconds. Runtimes do not include model initialization or I/O. Cost estimates are based on list prices for hardware given in Table 5. The CPU-only system used 40 Haswell CPU nodes. Four CPU-GPU configurations were used, where "numCPUs" indicate the total number of CPUs used, and "K80s per CPU" indicate the number of accelerators attached to each node.

Based on list prices in Table 3, a 40 node CPU would cost $260,000. Systems configured with 1 to 4 NVIDIA K80s per CPU are shown that lower the price of the system from $230K to $132.5K respectively. For these tests, 20 NVIDIA K80s were used containing 2 GPUs per board; no changes in run times were observed for the four CPU-GPU configurations. Systems such as Cray Storm support 8 or more K80s that could give additional cost benefit favoring GPUs.

6. Discussion

The NIM demonstrates that weather prediction codes can be designed for high performance and portability targeting CPU, GPU and MIC architectures with a single source code. Inherent in the design of NIM has been the simplicity of the code, use of basic Fortran language constructs, and minimal branching in loop calculations. Use of Fortran pointers, derived types, and other constructs that are not well supported or are challenging for compilers to analyze and optimize were avoided. NIM's icosahedral-hexagonal grid permits grid cells to be treated identically, which minimizes branching in grid point calculations. Further, code design separated fine-grain and coarse grain (MPI) parallelism. This was primarily due to limitations in F2C-ACC, but had a benefit of organizing calculations to avoid creation and execution of small parallel regions, where synchronization and thread startup (CPU, MIC) and synchronization or kernel startup (GPU) time can be significant.

[14] Each doubling in model resolution requires 4X more compute, and a 2X increase in the number of model time-steps. Assuming perfect scaling, an increase in model resolution from 30KM to 3.75KM requires 64X (4^3) more GPUs, with an 8X (2^3) increase in the number of model time-steps. Therefore, scaling to 3.75KM is calculated as 8 * 0.20 = 1.6% of real-time. Additional increases in compute and time-to-solution are expected when physics calculations are included.

The choice to organize arrays and loop calculations, with an innermost vertical dimension and indirect addressing to access neighboring grid cells, simplified code design without sacrificing performance. It also improved code portability and performance in unanticipated ways. First, the innermost vertical dimension of 96 levels was sufficient for CPU and MIC vectorization, but essential for the GPU's high-core count devices. With few dependencies in the vertical dimension, vectorization (CPU, MIC), and thread-parallelism (GPU) were consistently available in dynamics routines. Second, indirect addressing of grid cells gave flexibility and benefit in how they could be organized. For example, spiral grid re-ordering to eliminate MPI message packing and unpacking gave up to 20 percent improvement in model performance.

Optimizations benefitting one architecture also helped the others. In the rare event performance degraded on one or more architecture, the changes were re-formulated to gain positive benefit on all. The PGI OpenACC compiler now matches the performance of the F2C-ACC compiler. OpenACC compilers continue to mature, benefiting from F2C-ACC comparisons that exposed bugs and performance issues that were corrected. Parallelization is simpler with OpenACC, largely because data movement between CPU and GPU is managed by the runtime system. Unified memory on the GPU is expected to further simplify parallelization, narrowing the ease-of-use gap versus OpenMP. These improvements have led of a decision to phase out use of F2C-ACC, relying solely on OpenACC compilers for NIM and other models.

The scope of this paper primarily focused on model dynamics, largely because domain scientists had not decided which physics suite to use for high-resolution (< 4KM) runs. Parallelization of select microphysics and radiation routines improved performance on all architectures, but MIC and GPU runtimes were slower than the CPU (Henderson et al. 2015; Michalakes et al. 2015). Lower performance is likely due to more conditionals, and less parallelism in physics calculations than in dynamics routines. Conditionals restrict SIMD parallelism, limiting performance. Further parallelism is largely determined by data dependencies in the scientific formulations. Since physics routines generally contain dependencies in the vertical column, parallelism is often only available in the horizontal dimension. To ease this limitation, chunking was described, a means to divide parallelism between vectorization and thread parallelism on the CPU and MIC processors that can be similarly mapped to the thread and block parallelism on the GPU. Hyper-threading on the CPU and MIC was also described, an effective load balancing mechanism when multiple vertical columns are executed by each MPI task.

The paper gives a cost-benefit calculation for NIM dynamics that showed increasing value as more accelerators per node are used. However there are several limitations in the value of these results. First, the comparison was only for model dynamics; when physics is included, model performance and cost benefit favoring the GPU is expected to decrease. Second, use of list price is naïve as vendors typically offer significant discounts, particularly for large installations. Third, calculations did not include the cost of the system interconnect. For small systems with tens of nodes this was deemed acceptable for comparison as there would be little difference in price or performance. However, comparisons with hundreds to thousands of nodes would amplify the role of the interconnect and would need to be included in cost-benefit calculations.

Finally, the Next Generation Global Prediction System (NGGPS) program is tasked with developing the Nation's next global weather prediction model that will be used by NOAA's National Weather Service around 2020. As part of the High Impact Weather Prediction Project (HiWPP), two dynamical cores were selected as leading candidates to replace the existing operational Global Forecast System (GFS): NOAA GFDL's Finite-Volume cubed (FV3) (Lin 2004) and the Model Prediction Across Scales (MPAS) (Skamarock et al. 2015) from NCAR. The NIM work can be used as a template for MPFG parallelization of these models to enable running global forecast models operationally at 3KM resolution in the next decade.

7. Conclusion

The NIM is currently the only weather model able capable of running on CPU, GPU and MIC architectures with a single source code. Performance of the NIM dynamical core was described. CPU, GPU and MIC comparisons were made for device, node and multi-node performance. Performance calculations were made based on goal of running at 3KM resolution in 1-2 percent of real-time. Device comparisons show NIM ran on the MIC and GPU, 1.3X and 1.9X faster respectively than the same generation CPU hardware. The 1.3X MIC speedup versus a dual socket CPU is significant, since no other weather or climate model has been able to demonstrate speedup favoring the MIC. Single node comparisons demonstrated the value of symmetric execution where both the CPU and GPU or MIC hardware was used. For multi-node execution, the spiral grid ordering was described that eliminated data packing and unpacking and gave performance benefit on all architectures. Finally, a cost-benefit analysis demonstrated increasing benefits favoring the GPU when up to 8 accelerators were attached to each CPU host.

New hardware in 2016 looks promising and will invite further comparisons. Both GPU and MIC chips will include high speed memory with up to 5 times faster performance than current generation hardware. For the MIC, the hostless Knights-Landing processor will no longer be attached to a CPU host, which should also improve inter-node communications performance. For the GPU, the Pascal and Volta chips will offer hardware supported unified memory and faster communications which should improve programmability and performance. These improvements, along with more mature compilers, should further simplify porting codes to GPUs.

Fundamental to achieving good performance and portability has been the design of NIM. The simplicity of the code design, looping and array structures, and the indirect addressing of icosahedral grid were all chosen to expose the maximum parallelism to the underlying hardware. The work reported here, represents a successful development effort by a team of domain and computer scientists and software engineers. Scientists write the code, computer scientists are responsible for the directive-based parallelization and optimization, and software engineers maintain the software infrastructure capable of supporting development, testing and running the model on diverse supercomputer systems.

Applying NIM performance and portability techniques to the MPAS and the FV3 models is a next step in the work. The HiWPP report showed that both models demonstrated good scaling to 130,000 CPU cores (Michalakes et al., 2015). While these results indicate sufficient parallelism is available, some work will be required to adapt them to run efficiently on GPU and MIC processors.

In the next decade, HPC is expected to become increasingly fine-grained with systems containing potentially hundreds of millions of processing cores. To take advantage of these systems new weather prediction models will need to be co-developed by scientific and computational teams to incorporate parallelism in model design, code structure, algorithms, and underlying physical processes.

ACKNOWLEDGMENTS

Thanks to technical teams at Intel, Cray, PGI, and NVIDIA who were responsible for fixing bugs, and providing access to the latest hardware and compilers. Thanks also to the staff at ORNL Titan, and NSF TACC for providing system resources and helping to resolve system issues. This work was also supported in part by the Disaster Relief Appropriations Act of 2013 and the NOAA HPCC program.

REFERENCES

Arakawa, A., and V. R. Lamb, 1977: Computational design of the basic dynamical processes of the UCLA general circulation model. Meth. Comput. Phys., 17, Academic Press, New York, 173–265.

Bleck, R., J. Bao, S. Benjamin, J. Brown, M. Fiorino, T. Henderson, J. Lee, A. MacDonald, P. Madden, J. Middlecoff, J. Rosinski, T. Smirnova, S. Sun and N. Wang, 2015: *Monthly Weather Review*, 143, 2386-2403, DOI: 10.1175/MWR-D-14-00300.1

Carpenter I. R. Archibald, K. Evans, and M.Taylor, 2013: Progress towards accelerating HOMME on hybrid multi-core systems, *Intl Journal of High Performance Computing Applications*, 27(3), 335-347 – July 2013, DOI: 10.1177/1094342012462751.

Cirrascale, 2015: Scaling GPU compute performance, Cirrascale white paper, June 2015, http://www.cirrascale.com/documents/whitepapers/Cirrascale_ScalingGPUCompute_WP_M987_REVA.pdf.

CUDA C Programming Guide, 2015: http://docs.nvidia.com/cuda/cuda-c-programming-guide/.

Ellis, S., 2015: Exploring the PCI bus routines, CirraScale Blog post, August 13, 2014, http://www.cirrascale.com/blog/index.php/exploring-the-pcie-bus-routes/.

Fuhrer, O., C. Osuna, X. Lapillone, T. Gysi, B. Cumming, M. Bianco, A. Arteaga, and T. Schulthess, 2014: Towards a performance portable, architecture agnostic implementation strategy for weather and climate models, *Computing Frontiers and Innovations*, 1, No 1, DOI: 10.14529/jsfi1401.

Govett, M, L. Hart, T. Henderson, and D. Schaffer, 2003: The Scalable Modeling System: directive-based code parallelization for distributed and shared memory computers, *Parallel Computing*, 29(8), 995-1020.

Govett, M., J. Middlecoff and T. Henderson, 2010: Running the NIM next-generation weather model on GPUs, 10th IEEE/ACM Intl Conf on Cluster, Cloud and Grid Computing, CCGrid 2010, 17-20, May 2010, Melbourne, Australia.

Govett, M., Using OpenACC compilers to run FIM and NIM on GPUs, 2013 NCAR multi-core workshop, Boulder, Colorado, Sept 2013, https://www2.cisl.ucar.edu/sites/default/files/govett_6b.pdf

Govett, M., J. Middlecoff and T. Henderson, Directive-based parallelization of the NIM weather odel for GPUs, First Workshop on Accelerator Programming using Directives (WACCPD 14), pp 55-61, New Orleans, LA, November 2014.

Henderson, T., M. Govett, and J. Middlecoff, 2011: Applyng Fortran GPU compilers to numerical weather prediction, *2011 Symposium on Application Accelerators in High Performance Computing (SAAHPC 2011)*, pp 34-41, Knoxville, TN, August 2011.

Henderson, T., J. Michalkes, I. Gokhale and A.Jha, 2015: Optimizing Numerical Weather Prediction, *High Performance Parallelism Pearls Volume Two: Multicore and Many-core Programming Approaches*, 7-23, Morgan Kaufmann publisher, DOI: 10.1016/B978-0-12-803819-2.00016-1.

Lapillonne, X, and O.Fuhrer, 2014: Using Compiler Directives to Port Large Scientific Applications to GPUs: An Example from Atmospheric Science, ***Parallel Process. Letters*. 24**, 1450003 [18 pages] DOI: 10.1142/S0129626414500030.

Lee, J. and A.E. MacDonald, 2009: A finite-volume icosahedral shallow water model on local coordinate, *Monthly Weather Review*, 137, 1422-1437.

Lee, J., R. Bleck and A.E. MacDonald, 2010: A multistep flux corrected transport scheme, *Journal of Computing Physics*, 229, 9284-9298.

Lin, S-J. 2004: A vertically lagrangian finite-volume dynamical core for global models, J.Computing Physics, 229, 9284-9298.

MacDonald, A.E., J. Middlecoff, T. Henderson and J. Lee, 2011: A general method for modeling on irregular grids, *Intl Journal of High Performance Computing Applications*, 25(4), 392-403, DOI: http://dx.doi.org/10.1177/1094342010385019.

Michalakes, J., M. Iacono, and E. Jessup, 2015: Optimizing Weather Model Radiative Transfer Physics for Intel's Many Integrated Core (MIC) Architecture, preprint.

Michalakes, J, M. Govett, T. Black, H. Juang, A. Reinecke, and B. Skamarock, AVEC Report: NGGPS Level-1 Benchmarks and Software Evaluation, April 30, 2015, http://www.nws.noaa.gov/ost/nggps/DycoreTestingFiles/AVEC%20Level%201%20Benchmarking%20Report%2008%2020150602.pdf.

Middlecoff, J., 2015: Optimization of MPI message passing in a multi-core NWP dynamical core running on NVIDIA GPUs, in *Fifth NCAR multi-core workshop*, Boulder, Colorado, Sept 2015, https://www2.cisl.ucar.edu/sites/default/files/Abstract_Middlecoff.pdf.

Norman, M., I. Demeshko, J. Larkin, A. Vose, and M. Taylor, 2015: Experiences with CUDA and OpenACC from porting ACME to GPUs, in *Fifth NCAR multi-core workshop*, Boulder, Colorado, Sept 2015, https://www2.cisl.ucar.edu/sites/default/files/Norman_Slides.pdf.

Nyugen, H., C.Kerr, Z.Liang, 2013: Performance of the Cube-Sphere Atmospheric Dynamical Core on Xeon and Xeon-Phi Architectures, in *Third NCAR Multi-core Workshop,* Boulder Colorado, Sept 2013, https://www2.cisl.ucar.edu/sites/default/files/vu_3a.pdf.

Putnam, B., 2011: Graphics Processing Unit (GPU) acceleration of the Goddard Earth Observing System Atmospheric Model, *NASA Technical Report*, Jan 2011.

Rosinski, J., Porting and Optimizing NCEP's GFS Physics Package for Unstructured Grids on Intel Xeon and Xeon-Phi, in *Fifth NCAR Multi-core Workshop*, Boulder, Colorado, Sept 2015, https://www2.cisl.ucar.edu/sites/default/files/Rosinski_slides.pdf.

Sadourny, R., A. Arakawa and Y. Mintz, 1968: Integration of non-divergent barotropic vorticity equation with an icasahderal-hexagonal grid for the sphere, *Monthly Weather Review 96 (6): 351-356, DOI:* http://dx.doi.org/10.1175/1520-0493(1968)096<0351:IOTNBV>2.0.CO;2

Sawyer, W., Zaegal G., Linardakis,L., 2014: Towards a multi-node OpenACC Implementation of the ICON Model, EGU General Assembly 2014, held 27 April - 2 May, 2014 in Vienna, Austria, id.15276.

Skamarock, B., J. Klemp, M. Duda, L. Fowler, S. Park, and T. Ringler, 2012: A multi-scale non-hydrostatic atmospheric Model using centroidal voronoi tessellations and C-grid staggering, *Monthly Weather Review*, 240, 3090-3105, DOI 10.1175/MWR-D-11-00215.1.

Top500 Website, 2015: http://www.top500.org/lists/2015/11/.

Wang, N. and J. Lee, 2011: Geometric properties of the icosahedral-hexagonal grid on the two-sphere, SIAM J. Sci. Computing, 33(5): 2536-2559.

Whitaker, J., 2015: HIWPP non-hydrostatic dynamical core tests: Results from idealized test cases, Jan-21-2015, http://www.nws.noaa.gov/ost/nggps/DycoreTestingFiles/HIWPP_idealized_tests-v8%20revised%2005212015.pdf.

Williamson, D., 1971: A comparison of first and second-order difference approximations over a spherical geodesic grid, *J.Comp.Phys.*, 7(2),301-309, April 1971, DOI: doi:10.1016/0021-9991(71)90091-X.

Yashiro, H., A. Naruse, R. Yoshida, H. Tomita, 2014: A global atmosphere simulation on a GPU supercomputer using OpenACC: Dry dynamical cores tests, TSUBAME ESJ Vol.12 (Pub. Sep. 22, 2014).